ต้นฉบับไม่มีหน้า 143-144

No Page 143-144 In Original

รูปที่ 5
คำสั่งที่ระบบจัดการอาจห้ามไม่ให้ทาสก์ที่มีระดับความสำคัญต่ำใช้ ▶

```
IN          input from port
OUT         output to port
INS         Read an input string from port
OUTS        Write an output string to port
CLI         Disable interrupts
STI         Enable interrupts
```

```
HLT         Halt the processor
LGDT        Load the GDT register
LIDT        Load the interrupt-descriptor-table register
LLDT        Load the LDT register
CLTS        Clear the task-switched flag
LMSW        Load the MSW
LTR         Load the task register
```

รูปที่ 6 คำสั่งที่ใช้ได้เฉพาะในทาสก์ที่มีระดับความสำคัญสูงสุดเท่านั้น

ดำเนินการกับหน่วยความจำของทาสก์ที่มีระดับความสำคัญสูงกว่าไม่ได้

ทั้งนี้เพื่อให้การรันโปรแกรมในแบบมัลติทาสก์เป็นไปด้วยดี และไม่ทำให้โปรแกรมที่มีลำดับความสำคัญต่ำล้มการทำงานของระบบได้ เช่น การรันโปรแกรมที่อยู่ในระหว่างการทดลอง หากมีข้อผิดพลาดก็ให้สามารถกลับเข้าสู่ระบบจัดการได้ (แต่จะเห็นต่อไปว่าทำให้ผู้ใช้เข้าถึงซีพียู หน่วยความจำและพอร์ตได้ยากขึ้น ทุกอย่างต้องดำเนินการผ่านระบบจัดการ)

โหมดการทำงาน

แต่เนื่องจากอินเทลต้องการให้โปรแกรมที่เขียนสำหรับ 8086 ซึ่งมีอยู่เป็นจำนวนมากสามารถรันบน 80286 ได้ด้วย จึงได้แบ่งโหมดการทำงานของ 80286 เป็น 2 โหมด คือ Real Address Mode ซึ่งจะขอเรียกว่าเรียลโหมด เพื่อใช้ในการรันโปรแกรมที่เขียนสำหรับ 8086/88 เดิม ที่เรียกว่าเป็น Real Mode เพราะใช้วิธีกำหนดแอดเดรสโดยตรงกับหน่วยความจำ

อีกโหมดการทำงานหนึ่งคือ Protected Virtual Address Mode ซึ่งจะขอเรียกว่าโปรเทกโหมด จุดประสงค์เพื่อใช้กับโปรแกรมที่เขียนใหม่ โดยเฉพาะระบบจัดการที่เขียนใหม่ในแบบมัลติทาสก์ที่เรียกว่าเป็น Protected เพราะมีการป้องกันการทำงานของทาสก์ดังกล่าวแล้ว และที่เรียกว่าเป็น Virtual Address เพราะมีการอ้างแอดเดรสเทียมดังจะกล่าวต่อไป

รีจิสเตอร์

เพื่อที่จะใช้ในการอ้างอิงต่อไปจะขอนำรีจิสเตอร์ใน 80286 มากล่าวถึงก่อน โดยจะกล่าวถึงเฉพาะชื่อกับขนาด ส่วนหน้าที่การทำงานจะกล่าวรายละเอียดในแต่ละเรื่องต่อไป เราสามารถแบ่งรีจิสเตอร์ใน 80286 ได้เป็น 3 กลุ่ม คือ

General Purpose Registers

เป็นรีจิสเตอร์ตามแบบของ 8086 (จึงจะไม่กล่าวในรายละเอียด) รีจิสเตอร์พวกนี้เป็นรีจิสเตอร์ขนาด 16 บิต สามารถแบ่งออกได้เป็น 4 กลุ่มย่อย คือ

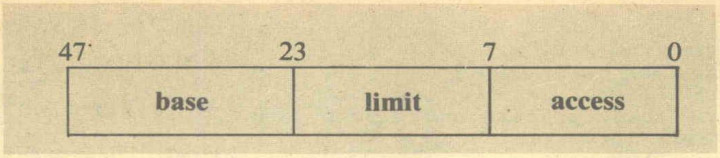
- **Data** เป็นรีจิสเตอร์สำหรับเก็บข้อมูลและดำเนินกรรมวิธีข้อมูล ได้แก่AX, BX, CX และ DX ซึ่งแต่ละตัวสามารถแยกออกเป็นรีจิสเตอร์ขนาด 8 บิต 2 ตัว คือเป็น AH, AL, BH, BL, CH, CL, DH และ DL
- **Address** เป็นรีจิสเตอร์ที่ใช้ในการอ้างแอดเดรส คือ BX, SI, DI, BP และ SP (BX เป็นทั้ง Data และ Address) กับ IP ซึ่งใช้ในการชี้คำสั่ง
- **Segment** เป็นรีจิสเตอร์ที่ใช้ในการอ้างแอดเดรสเช่นกันแต่อ้างเป็นเซกเมนต์ เซกเมนต์ละ 64 กิโลไบต์ ได้แก่ CS, DS, ES และ SS
- **Flags** คือรีจิสเตอร์ Flags เก็บแฟล็ก (flag) แสดงสถานะการทำงานของซีพียู

System Registers

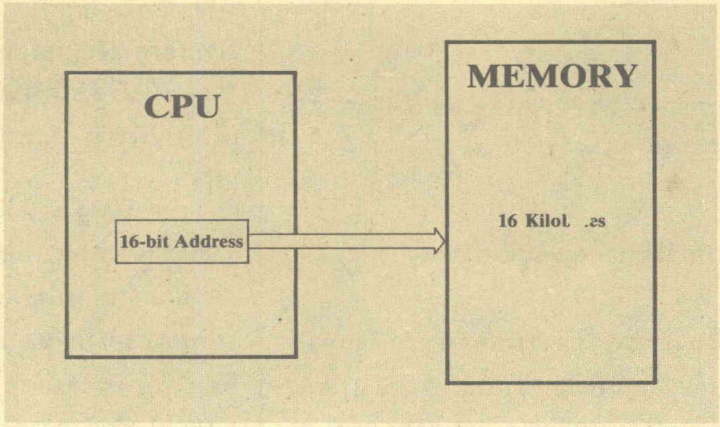
เป็นรีจิสเตอร์ที่กำหนดขึ้นใหม่เพื่อการทำงานในแบบโปรเทกโหมดได้แก่

- **GDTR** (Global Descriptor Table Register) เป็นรีจิสเตอร์ขนาด 40 บิต แบ่งเป็น 2 ท่อน ท่อนบน (บิตสูง) ขนาด 24 บิต เรียกว่า GDT base เป็นพอยน์เตอร์ชี้ไปยังต้นตาราง GDT นั่นคือเราสามารถกำหนดต้นตาราง GDT ไว้ตรงไหนก็ได้ของหน่วยความจำขนาด 16 เมกะไบต์ (ตามที่ขาแอดเดรสของซีพียู่ 24 ขา สามารถอ้างแอดเดรสได้) ท่อนล่างขนาด 16 บิต เรียกว่า GDT limit เป็นค่าออฟเซตจาก GDT base ชี้ไปยังท้ายตาราง นั่นคือเนื้อที่ของตาราง GDT จะมีได้ไม่เกิน 64 กิโลไบต์
- IDTR (Interrupt Descriptor Table Register) มีขนาดและหน้าที่เช่นเดียวกับ GDTR แต่ชี้ไปต้นตาราง IDT
- LDTR (Local Descriptor Table Register) เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งเป็นออฟเซตจาก GDT base ชี้ไปยัง LDT Descriptor ใน GDT เพื่ออ่านค่ามาเป็นพอยน์เตอร์ชี้ไปยังต้นตาราง LDT อีกทอดหนึ่ง (GDTR และ LDTR ตั้งชื่อไว้ทำนองเดียวกันแต่ทำหน้าที่ต่างกัน)
- TR (Task Register) เป็นรีจิสเตอร์ขนาด 16 บิต ซึ่งเป็นออฟเซตจาก GDT base ชี้ไปยัง TSS Descriptor ใน GDT เพื่ออ่านค่ามาเป็นพอยน์เตอร์ชี้ไปยังต้น TSS คือ Task State Segment อีกทอดหนึ่ง
- MSW (Machine Status Word) เป็นรีจิสเตอร์ขนาด 16 บิต

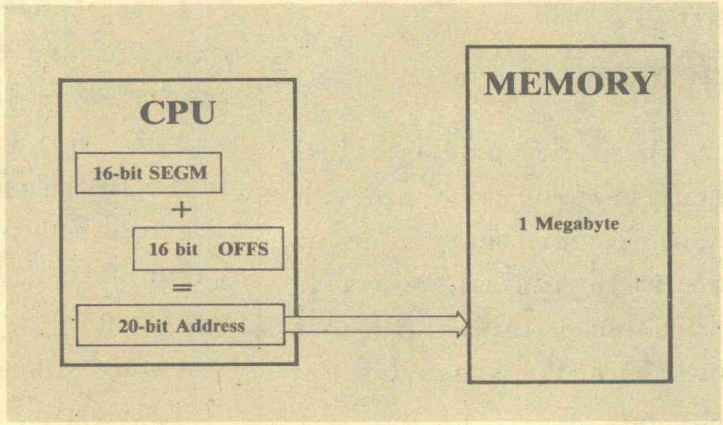
รูปที่ 7 ▶ แสดงหน่วยความจำ descriptor cache ใน CPU ซึ่ง cache หนึ่ง ๆ จะมีขนาด 48 ไบต์ แบ่งเป็น base, limit และ access

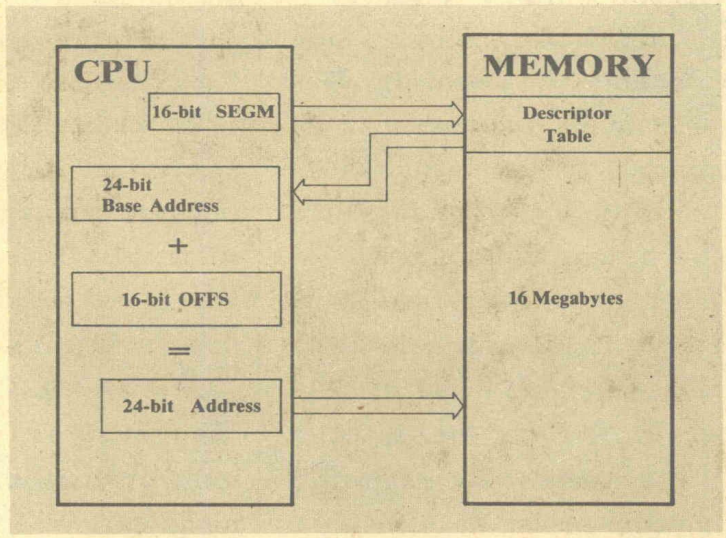


รูปที่ 8 ▶ การอ้างแอดเดรสในซีพียู 8 บิต



รูปที่ 9 การอ้างแอดเดรสของ 8086/8088 นำค่าเซกเมนต์มาบวกกับออฟเซตในแบบเหลื่อมบิต เป็นค่าแอดเดรสให้แก่หน่วยความจำ





รูปที่ 10 การอ้างแอดเดรสของ 80286 ดำเนินการโดยซีพียูนำค่าเซกเมนต์ (ซึ่งเปลี่ยนเรียกเป็นซีเลคเตอร์) ไปเปิดตาราง Descriptor Table ได้ค่า Bass Address นำมาบวกกับค่าออฟเซตได้เป็นแอดเดรส ถึงหน่วยความจำ

แต่ใช้งานจริงเพียง 4 บิตต่ำทำหน้าที่เป็นแฟล็ก

Caches

ความจริงแล้ว Caches เป็นหน่วยความจำในซีพียู แต่ทำหน้าที่ของรีจิสเตอร์จึงจัดอยู่ในพวกรีจิสเตอร์ด้วย มีทั้งหมด 6 ตัวด้วยกันคือ

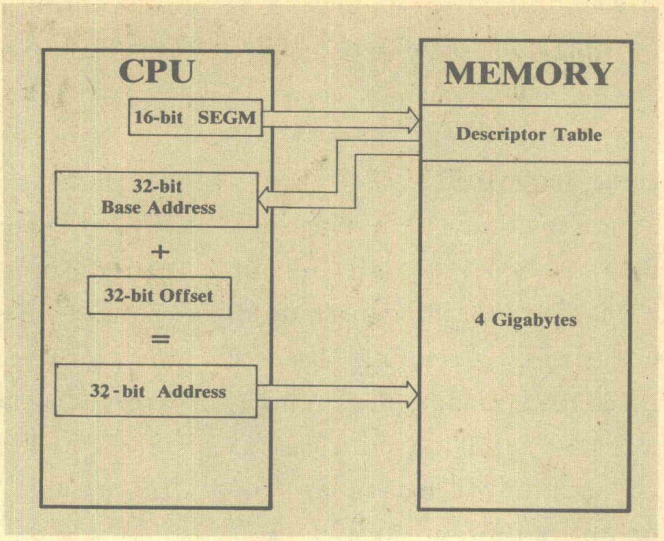
- CS descriptor cache
- DS descriptor cache
- SS descriptor cache
- ES descriptor cache
- LDTR descriptor cache
- TR descriptor cache

รีจิสเตอร์พวกนี้มีขนาด 48 บิต แบ่งออกเป็น 3 ตอน ดังแสดงในรูปที่ 7 โดยที่ base คือพอยน์เตอร์ เช่นถ้าเป็น base ของ CS cache ก็จะชี้ไปที่ต้น code segment ส่วน limit เป็นค่าออฟเซตจาก base ชี้ไปที่ท้ายเซกเมนต์ สำหรับ access ย่อมาจาก access right คือสิทธิในการเข้าถึงหน่วยความจำของทาสก์ของ CS นี้ (จะกล่าวในรายละเอียดต่อไป)

รูปแบบของการกำหนดแอดเดรส

ได้แสดงรูปแบบของการกำหนดแอดเดรสถึงหน่วยความจำของซีพียูเบอร์ต่าง ๆ ของอินเทลไว้ในรูปที่ 8 ถึง 11 เพื่อเปรียบเทียบกัน คือ

- ในยุค 8 บิต คือ 8080 การกำหนดแอดเดรสกระทำโดยตรงต่อหน่วยความจำได้ไม่เกิน 64 กิโลไบต์ (ขาแอดเดรส 16 ขา)
- พอขึ้นยุค 16 บิต คือ 8086 อินเทลต้องการให้อ้างแอดเดรสได้ถึง 1 เมกะไบต์ จึงได้จัดขาแอดเดรสไว้ 20 ขา แต่ภายในซีพียูไม่มีรีจิสเตอร์ตัวใดเลยที่มีขนาดเกิน 16 บิต อินเทลแก้ปัญหานี้ด้วยการให้อ้างแอดเดรสเป็นเซกเมนต์ : ออฟเซต โดยนำค่าทั้งสองนี้มาบวกกัน (ความจริงเป็นการต่อกัน)



รูปที่ 11
การอ้างแอดเดรสของ 80386 ดำเนินการเช่นเดียวกับของ 80286 แต่เนื่องจากใช้รีจิสเตอร์ขนาด 32 บิต จึงอ้างแอดเดรสได้เซกเมนต์ละ 4 กิ๊กกะไบต์

ในแบบเหลื่อมบิต แต่ก็ยังคงเป็นการกำหนดแอดเดรสโดยตรงเพราะดำเนินการภายในซีพียู

● เมื่อพัฒนาเป็น 80286 อินเทลต้องการให้เพิ่มหน่วยความจำได้อีก จึงได้จัดขาแอดเดรสไว้ 24 ขา สามารถจัดหน่วยความจำได้ถึง 16 เมกะไบต์ และเพื่อการทำงานในโปรเทกโหมดจึงได้จัดรีจิสเตอร์ดังที่ได้กล่าวมาแล้ว การอ้างแอดเดรสของผู้เขียนโปรแกรมยังคงเป็นแบบเดิมคือเป็น เซกเมนต์ : ออฟเซต แต่การกำหนดแอดเดรสของซีพียูเป็นการดำเนินการทางอ้อมคือ ซีพียูจะนำค่าเซกเมนต์ไปเปิดตาราง Descriptor ได้ค่าเบสแอดเดรสมา นำมาบวกกับค่าออฟเซต เป็นแอดเดรสถึงหน่วยความจำ

● ขึ้นยุค 32 บิต คือ 80386 อินเทลยังคงใช้โครงสร้างภายในตามเดิม แต่ได้เพิ่มขาแอดเดรสขึ้นเป็น 32 ขา ทำให้สามารถอ้างแอดเดรสได้ถึง 4 กิ๊กกะไบต์ ส่วนรีจิสเตอร์เว้นพวกเซกเมนต์ได้เพิ่มเป็น 32 บิต และได้เพิ่มค่าออฟเซตเป็น 32 บิตด้วย จึงทำให้สามารถอ้างแอดเดรสต่อเนื่องกันได้เซกเมนต์ละ 4 กิกะไบต์ แทนของเดิมซึ่งอ้างต่อเนื่องได้เพียงเซกเมนต์ละ 64 กิโลไบต์ เพราะมีค่าออฟเซตเพียง 16 บิต

การกำหนดแอดเดรสในโปรเทกโหมด

เพื่อให้สามารถกำหนดแอดเดรสได้ดังกล่าวมา อินเทลได้ออกแบบให้สร้างตารางหลายตาราง ตารางแรกที่จะนำมากล่าวถึง คือ ตาราง GDT

ในการอธิบายการทำงานหากเรื่องใดที่ซีพียูดำเนินการโดยอัตโนมัติ ก็จะกล่าวว่าเป็นการดำเนินการของซีพียู หากเรื่องใดต้องเขียนเป็นโปรแกรม เพื่อความเข้าใจง่ายจะขอแบ่งเป็น 2 ระดับคือ เมื่อกล่าวว่าโปรแกรมระบบจัดการเป็นผู้ดำเนินการขอให้หมายถึงผู้เขียนโปรแกรมระบบจัดการ เมื่อกล่าวว่าโปรแกรมใช้งานเป็นผู้ดำเนินการขอให้หมายถึงผู้เขียนโปรแกรมใช้งาน อนึ่งคำอธิบายในหัวข้อนี้ยังไม่ใช่เรื่องของมัลติทาสก์โดยตรงครับ

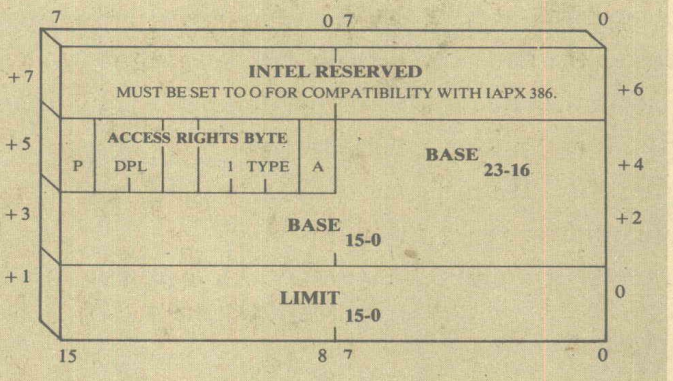
ตาราง GDT (Global Descriptor Table)

หลังจากรีเซตแล้ว 80286 จะทำงานในเรียลโหมด เมื่อต้องการให้ซีพียูทำงานในโปรเทกโหมดระบบจัดการต้องเตรียมตาราง GDT ไว้ให้พร้อมก่อนที่จะเซตบิต 0 ใน MSW เพื่อให้ซีพียูทำงานในโปรเทกโหมด

ตาราง GDT ก็คือตารางของเดสคริพเตอร์ (Descriptor) โดยที่เดสคริพเตอร์หนึ่ง ๆ จะใช้หน่วยความจำ 8 ไบต์ ดังแสดงในรูปที่ 12 ในตารางนี้จะมีกี่เดสคริพเตอร์ก็ได้ขึ้นอยู่กับจุดประสงค์ในการออกแบบเครื่องและระบบจัดการ แต่จะถูกจำกัดด้วยเนื้อที่ของตาราง GDT ซึ่งมีอยู่ 64 กิโลไบต์ นั่นคือจะมีเดสคริพเตอร์ได้ไม่เกิน 64K ÷ 8 = 8192 ตัว

เดสคริพเตอร์กับทาสก์

ตามที่กล่าวว่าก่อนที่จะให้ซีพียูทำงานในโปรเทกโหมดต้องเตรียมตาราง



รูปที่ 12
แสดงการจัดหน่วยความจำเป็นเดสคริพเตอร์ โดยที่ไบต์ 6 และ 7 สำรองไว้สำหรับ 80386 ไบต์ 5 คือ Access Rights Byte ไบต์ 2 ถึง 4 เก็บค่าเบส ไบต์ 0 และ 1 เก็บค่าลิมิต

GDT ไว้ให้พร้อมนั้น จะต้องให้ค่าแอดเดรสของต้นตาราง GDT และค่าลิมิต (limit) แก่รีจิสเตอร์ GDTR ก่อนด้วย (ยังมีเรื่องอื่นอีกครับ) ซึ่งจะให้เพียงครั้งแรกครั้งเดียว แต่ค่าในเดสคริพเตอร์แต่ละตัวนั้นเปลี่ยนแปลงได้และเปลี่ยนแปลงอยู่ตลอดเวลา

จากนี้ก็สามารถใช้ตาราง GDT ได้ โดยที่เดสคริพเตอร์หนึ่ง ๆ จะต้องเป็นของทาสก์ใดทาสก์หนึ่งเท่านั้น แต่ทาสก์หนึ่งอาจมีหลายเดสคริพเตอร์ เช่น ในโปรแกรมใช้งานตามรูปที่ 13 ระบบจัดการจะต้องจัดเดสคริพเตอร์หนึ่งให้แก่ CS และอีกเดสคริพเตอร์หนึ่งให้แก่ DS ของโปรแกรมหรือของทาสก์นี้

การดำเนินการจะเป็นดังนี้ ซึ่งจะขอกล่าวเฉพาะ DS คือ เมื่อระบบจัดการโหลดโปรแกรมนี้ลงหน่วยความจำ จะกำหนดเดสคริพเตอร์ที่ว่างให้แก่ DS ตรวจดูว่าโปรแกรมนี้ต้องการเนื้อที่ในหน่วยความจำสำหรับข้อมูลเท่าไร และมีเนื้อที่ในหน่วยความจำตอนใดว่างสำหรับข้อมูลนี้ นำค่าแอดเดรสของหน่วยความจำนี้ไปให้แก่เบส (base) และขนาดของข้อมูลลบหนึ่งไปให้แก่ลิมิต(limit) ของเดสคริพเตอร์นี้ (สำหรับ access rights จะกล่าวภายหลัง)

จากนี้จะให้ค่าออฟเซตของเดสคริพเตอร์นี้จากต้นตารางแก่ Dseg (ซึ่งถือว่าเป็นตัวแปร) แต่ค่าออฟเซตนี้เรียกเป็นซีเลคเตอร์(selector)

การรันโปรแกรมนี้จะเป็นการให้ค่าซีเลคเตอร์ใน Desg แก่ AX และนำมาถ่ายทอดให้แก่ DS อีกทอดหนึ่ง ซึ่งเมื่อให้ค่าซีเลคเตอร์แก่ DS ซีพียูจะโหลดค่า base, limit และ access ในเดสคริพเตอร์นี้ให้แก่รีจิสเตอร์ DS cache โดยอัตโนมัติ

ซึ่งตั้งแต่นี้ต่อไปเมื่อเราอ้างแอดเดรส เช่น อ้างชื่อตัวแปร ก็จะเป็นการอ้างถึงหน่วยความจำในเซกเมนต์นี้ แอดเดรสที่ซีพียูอ้างจะเป็นค่า base (ใน DS cache) บวกด้วยออฟเซต เช่น ตัวแปร base จะมีแอดเดรสเป็น base+1

จะเห็นว่าค่า base ก็คือค่าแอดเดรสต้นเซกเมนต์คล้ายกับที่ระบบจัดการเคยให้แก่ DS ใน 8086 แต่เป็นขนาด 24 บิต จึงต้องให้แก่ DS cache สำหรับ limit ก็เพื่อให้ซีพียูควบคุมการทำงานของโปรแกรมนี้ให้อยู่ในเซกเมนต์ที่กำหนด

การดำเนินการเช่นนี้จะทำให้ผู้เขียนโปรแกรมใช้งานไม่รู้สึกถึงความแตกต่างเลยว่าเขียนโปรแกรมภาษาเครื่องสำหรับ 8086 หรือ 80286 แต่ที่กล่าวว่าโปรแกรมบางโปรแกรมที่เขียนสำหรับ 8086/88 เดิมใช้ไม่ได้กับ 80286 คือโปรแกรมที่อ้างแอดเดรสโดยตรง เช่น

MOV AX, 0B800h
MOV ES,AX

เป็นการเตรียม ES สำหรับอ้างถึงหน่วยความจำที่เป็นจอภาพสีโดยตรง แต่ในโปรเทกโหมดค่า B800h ไม่ใช่ค่าเซกเมนต์อีกต่อไป หากเป็นค่าซีเลคเตอร์จึงใช้ไม่ได้ นั่นคือในโปรเทกโหมดการดำเนินกับคีย์บอร์ด จอภาพ และอุปกรณ์อื่น ๆ ต้องดำเนินการผ่านระบบจัดการเท่านั้น

จากที่กล่าวมานี้เราสามารถสร้างตารางเดสคริพเตอร์ได้ถึง 8192 เดสคริพเตอร์ หรือกล่าวอีกนัยหนึ่งคือ เราสามารถกำหนดเซกเมนต์ได้ถึง 8192 เซกเมนต์ ซึ่งเซกเมนต์หนึ่ง ๆ อาจมีขนาดได้ถึง 64 กิโลไบต์ นั่นคือเราสามารถอ้างแอดเดรสในโปรแกรมได้ถึง 8192 × 64 กิโลไบต์ เท่ากับครึ่งกิ๊กกะไบต์

แต่เป็นการอ้างแอดเดรสไปยังหน่วยความจำที่ไม่มีตัวตนจริง(Virtual memory) แอดเดรสถึงหน่วยความจำจริงระบบจัดการเป็นผู้กำหนดได้แก่ base ซึ่งจะซ้ำกันทำให้มีการโหลดและเซฟกับดิสค์ตลอดเวลา วิธีการเช่นนี้เป็นการเพิ่มสมรรถภาพแก่ซีพียูอย่างหนึ่งคือ ให้สามารถอ้างแอดเดรสได้เป็นจำนวนมาก แต่มีหน่วยความจำจริงน้อย ซึ่งเป็นเรื่องสำคัญอันหนึ่งในการทำงานในแบบมัลติทาสก์

รูปที่ 13
โปรแกรม
ประกอบคำอธิบาย

```
dseg    SEGMENT
char    db    ?
dseg    ENDS

cseg    SEGMENT
        ASSUME  CS:cseg,  DS:dseg
mov     ax,dseg
mov     ds,ax
mov     char,'A'
cseg    ENDS
```

ตาราง LDT (Local Descriptor Table)

ในโปรแกรมที่ใช้ CS, DS, ES และอาจรวมถึง SS ด้วย หลายเซกเมนต์ เช่น โปรแกรมที่มี code หรือ data หรือทั้งคู่เกินกว่า 64 กิโลไบต์ ระบบจัดการจะจัดตาราง LDT ให้สำหรับโปรแกรม(ทาสก์)นี้โดยเฉพาะ และอาจมองเป็นได้ว่าโปรแกรมหรือทาสก์นี้ประกอบด้วยทาสก์ย่อยหลาย ๆ ทาสก์

การดำเนินการของระบบจัดการจะเริ่มด้วยการจัดทำตาราง LDT เช่นเดียวกับที่จัดทำตาราง GDT กำหนดเดสคริพเตอร์หนึ่งในตาราง GDT เพื่อการนี้ (ขอย้ำครับว่าเป็นเดสคริพเตอร์ในตาราง GDT คือที่กล่าวมาแล้ว) เพื่อความสะดวกในการอธิบายขอ

กำหนดเรียกเดสคริพเตอร์นี้เป็น Lima

เสร็จแล้วระบบจัดการจะต้องนำแอดเดรสต้นของตาราง LDT ไปให้เป็นค่า base และนำจำนวนเดสคริพเตอร์คูณด้วยแปดลบด้วยหนึ่งไปให้เป็นค่า limit รวมทั้งค่า access right ด้วย (ซึ่งยังไม่ได้กล่าวถึง) ของเดสคริพเตอร์ lima

จากนี้ระบบจัดการจะต้องโหลดค่าซีเลคเตอร์ของเดสคริพเตอร์ Lima ให้แก่รีจิสเตอร์ LDTR (ขนาด 16 บิต) ซึ่งซีพียูจะโหลดค่า base, limit และ access จากเดสคริพเตอร์ Lima ให้แก่ LDTR cache โดยอัตโนมัติ

ต่อไปในการโหลดค่าให้แก่ cashe ของ CS หรือ DS หรือ ES หรือ SS (ซึ่งได้กล่าวแล้วว่าทำโดยให้ค่าซีเลคเตอร์แก่รีจิสเตอร์เหล่านี้ แล้วซีพียูจะโหลดค่าจากเดสคริพเตอร์ให้แก่ cashe โดยอัตโนมัตินั้น) ก็จะเกิดปัญหาว่าจากตารางไหน GDT หรือ LDT อินเทลแก้ปัญหานี้โดยนำ 3 บิตต่ำของซีเลคเตอร์ซึ่งมีค่าเป็น 0 ตลอดเวลามาใช้งาน โดยถ้าให้บิต 2 เป็น 0 หมายถึง GDT ถ้าเป็น 1 หมายถึง LDT

เช่นเดียวกับ GDT เราสามารถจะจัดตาราง LDT ได้ถึง 8192 เดสคริพเตอร์ นั่นคือสามารถอ้างแอดเดรสเทียมได้อีกครึ่งกิ๊กกะไบต์ รวมทั้งหมด 80286 สามารถอ้างแอดเดรสเทียบได้ถึง 1 กิ๊กกะไบต์

Access Rights Byte

มีข้อมูลอีก 1 ไบต์ ในเดสคริพเตอร์ที่ยังไม่ได้กล่าวถึงคือ Access Rights Byte ซึ่งบางทีอาจเรียกว่า Access Rigits หรือ Access เฉยๆ ความหมายของชื่อนี้คือสิทธิในการเข้าถึงหน่วยความจำ แต่มีเรื่องอื่นๆ อยู่ด้วยดังแสดงในรูปที่ 14 ซึ่งมีรายละเอียดดังนี้

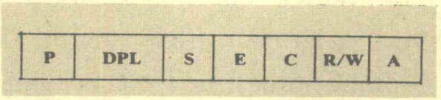
● **E** หมายถึง Executable คือเซกเมนต์นี้เป็น CS เมื่อให้ค่าเป็น 1 และเป็น data เมื่อให้ค่าเป็น 0

● **P** หมายถึง Present คือมีข้อมูลหรือโปรแกรมอยู่ในหน่วยความจำแล้วเมื่อมีค่าเป็น 1 หากเป็น 0 แสดงว่ายังไม่ได้โหลดลงหน่วยความจำ หากรันโปรแกรมหรือใช้ข้อมูลในเซกเมนต์นี้ ซีพียูจะอินเตอร์รัพต์ นั่นคือทำให้ระบบจัดการไม่จำเป็นต้องโหลดโปรแกรมหรือข้อมูลมาคอย

● **DPL** หมายถึง Descriptor Priviledge Level คือ ระดับความสำคัญของเซกเมนต์นี้ตามที่กล่าวในตอนต้นว่ามี 4 ระดับ จึงใช้หน่วยความจำ 2 บิต ได้ค่าเป็น level 0 (สูงสุด) ถึง level 3 (ต่ำสุด)

● **S** หมายถึง Segment คือ ถ้าเดสคริพเตอร์นี้แทนเซกเมนต์จะให้ค่าเป็น 1 แต่ถ้าชี้ไปที่ต้นตารางใด ๆ ก็ตาม เช่น เดสคริพเตอร์ Lima จะให้ค่าเป็น 0 เมื่อเป็น 0 ความหมายของ 3 บิตต่ำที่จะกล่าวต่อไปนี้เปลี่ยนไปด้วย (ความหมายที่เปลี่ยนใหม่จะกล่าวในแต่ละเรื่องต่อไป)

● **C** หมายถึง Conforming (ขอเรียนให้ทราบก่อนว่าหน้าที่ของบิตนี้ผมอ่านแล้วไม่ค่อยเข้าใจพอจับความได้ว่า) ถ้าให้ค่าเป็น 1 จะยินยอมให้โปรแกรมทุกระดับความสำคัญ เรียกใช้โปรแกรมย่อยในเซกเมนต์นี้ได้ ถ้าเป็น 0 ต้องเป็นไปตามลำดับความสำคัญ



รูปที่ 14 แสดงความหมายของแต่ละบิตใน Access Rights Byte

● **R/W** หมายถึง Read/Write กล่าวคือ ถ้าเป็น 0 หมายถึง ห้าม Read โปรแกรมในฐานะข้อมูล (ห้ามลอกโปรแกรม) หรือห้าม Write ข้อมูล แล้วแต่ว่าเป็น code segment หรือเป็น data segment ถ้าเป็น 1 ให้ Read โปรแกรม หรือ Write ข้อมูลได้

● **A** หมายถึง Access (บิตนี้เช่นกับครับ ไม่ค่อยเข้าใจ) ถ้าเป็น 1 ให้ดำเนินการกับเซกเมนต์นี้ได้ ถ้าเป็น 0 ห้ามดำเนินการ

อินเทอร์รัพต์เวคเตอร์ (Interrupt vectors)

ใน 8086 ได้กำหนดหน่วยความจำที่เป็นอินเตอร์รัพต์เวคเตอร์ไว้แน่นอน คือ ตั้งแต่แอดเดรส 0000 : 0000 ถึงแอดเดรส 0000 : 03FF แต่ใน 80286 ระบบจัดการจะกำหนดอินเตอร์รัพต์เวคเตอร์ไว้ ณ ที่ใดก็ได้ โดยทำเป็นตาราง IDT (Interrupt Descriptor Table) แล้วให้ค่าแอดเดรสของต้นตาราง ค่า limit และ Access แก่รีจิสเตอร์ IDTR

ขอให้สังเกตด้วยว่ารีจิสเตอร์ IDTR ชี้ไปยังต้นตาราง IDT โดยตรง เช่นเดียวกับที่ GDTR ชี้ไปยังต้นตาราง GDT ไม่เหมือนกับรีจิสเตอร์ LDTR ที่ต้องชี้ผ่านเดสคริพเตอร์ในตาราง GDT

สำหรับแบบของเดสคริพเตอร์ในตาราง IDT เช่นเดียวกับของ GDT คือประกอบด้วย base, limit, access และไบต์ 0 (สำหรับ 80386) แต่ตาราง IDT มีขนาดเล็กกว่า GDT เนื่องจากใน 80386 (เช่นเดียวกับ 8086) มีอินเตอร์รัพต์เวคเตอร์เพียง 256 เวคเตอร์ จึงไม่มีความจำเป็นใดที่จะกำหนดเดสคริพเตอร์เกินกว่า 256 เดสคริพเตอร์

ดังได้กล่าวแล้วว่า 80286 ทำงานได้ 2 โหมด คือ เรียลโหมดกับโปรเทกโหมด ตาราง IDT ที่กล่าวมานี้สำหรับการรันในแบบโปรเทกโหมด หากรันในเรียลโหมดคงใช้ตารางอินเตอร์รัพต์เวคเตอร์ตามแบบของ 8086 นั่นคือระบบจัดการที่เขียนสำหรับ 8086 สามารถใช้กับ 80286 ได้โดยตรง

แต่ในความเป็นจริงนั้นซีพียูจะนำหมายเลขอินเตอร์รัพต์คูณด้วย 4 เป็นออฟเซตจาก IDTR ซึ่งถูกให้ค่าเป็น 0 เมื่อรีเซต จึงตรงกับ 8086 พอดี แต่ระบบจัดการที่เขียนใหม่สำหรับการรันในแบบเรียลโหมดอาจย้ายตารางอินเตอร์รัพต์เวคเตอร์ไปไว้ที่ใดก็ได้โดยให้ค่า IDTR ใหม่

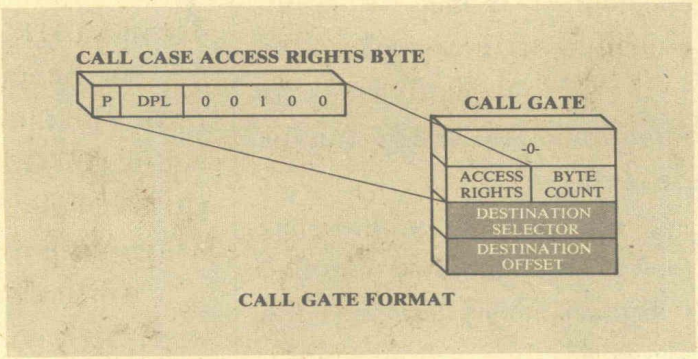
อนึ่งเนื่องจากค่าแอดเดรสของโปรแกรมสนองตอบการอินเตอร์รัพต์ (interrupt service routine) ที่ให้ไว้ในตารางทั้งสองนี้ไม่เหมือนกัน กล่าวคือในตารางอินเตอร์รัพต์เวคเตอร์เป็นแอดเดรส 32 บิต แบบเซกเมนต์ : ออฟเซต ส่วนในตาราง IDT เป็นแอดเดรสโดยตรง 24 บิต จึงกำหนดเรียกให้แตกต่างกันว่าเกต(gate) สำหรับหมายเลขอินเตอร์รัพต์คงให้ตรงกัน เช่น Vector 0 กับ Gate 0 ต่างก็เป็น Divide error ด้วยกันทั้งคู่

การเรียกเข้าหาระบบจัดการ

เรื่องราวที่ผมกล่าวมาทั้งหมดรวมทั้งที่จะกล่าวในหัวข้อนี้ด้วย ถึงแม้จะกล่าวถึงการทำงานในโปรเทกโหมด แต่ก็ยังคงเป็นการทำงานในแบบซิงเกิลทาสก์(single task) คือที่ระบบจัดการรันโปรแกรมใช้งานเพียงครั้งละหนึ่งเดียว

แต่สำหรับใน 80286 ระบบจัดการอาจโหลดโปรแกรมใช้งานลงหน่วยความจำหลายโปรแกรมหรือหลายทาสก์ อีกกรณีหนึ่งคือโปรแกรมขนาดใหญ่อาจแบ่งออกเป็นหลายทาสก์ย่อย การเรียกใช้โปรแกรมย่อยต่างทาสก์กันนี้ หากมีระดับความสำคัญเท่ากันจะเรียกใช้กันได้โดยตรงเช่นเดียวกับใน 8086

แต่ถ้าต่างระดับความสำคัญกันโดยที่โปรแกรมผู้เรียกมีระดับความสำคัญต่ำกว่าเช่น ให้โปรแกรมใช้งานเรียกโปรแกรมย่อยในระบบจัดการ จะเรียกโดยตรงไม่ได้ (เกี่ยวกับการป้องกันดังกล่าวมาแล้ว) แต่ความจำเป็นที่จะให้เรียกได้มีมากพอสมควร อินเทลจึงให้ทำได้โดยให้เป็นการเรียกผ่านเกต (gate) ซึ่งหมายถึงประตู (การเรียกโปรแกรมย่อยผ่านเกตนี้เป็นคนละเรื่องกับซอฟต์แวร์อินเตอร์รัพต์ แต่จะทำให้ลดการใช้ซอฟต์แวร์อินเตอร์รัพต์ลง)



รูปที่ 15
แสดงถึงลักษณะของคอลล์เกตเดสคริพเตอร์

คอลล์เกต(Callgate)

เกตในกรณีนี้เป็นคนละเรื่องกับเกตที่กล่าวในเรื่องอินเตอร์รัพต์เวคเตอร์ คือจะเป็นเดสคริพเตอร์ตัวหนึ่งในตาราง GDT สำหรับเกตหนึ่ง ๆ เรียกว่าคอลล์เกตเดสคริพเตอร์ (Call gate descriptor) เพราะมีลักษณะแตกต่างไปกับเดสคริพเตอร์ตามปกติ ดังแสดงในรูปที่ 15 กล่าวคือ

- **Access rights byte** จะมีรหัสพิเศษดังแสดงทำให้ซีพียูทราบว่าเป็นคอลล์เกตเดสคริพเตอร์
- **Destination selector** จะเป็นซีเลคเตอร์เพื่อชี้ไปยังเดสคริพเตอร์ธรรมดาอีกตัวหนึ่ง ซึ่งก็คือเดสคริพเตอร์ของทาสก์หรือคือเซกเมนต์ที่มีโปรแกรมย่อยที่ต้องการนั่นเอง
- **Destination offset** เป็นค่าออฟเซตในเซกเมนต์ดังกล่าวไปยังโปรแกรมย่อยที่ต้องการ
- **Byte count** จำนวนไบต์ของข้อมูลในสแต็กของโปรแกรมผู้เรียกที่จะถ่ายทอดไปให้แก่สแต็กของโปรแกรมที่ถูกเรียก ซึ่งสามารถให้ค่าได้สูงสุด 31 ไบต์ (ใช้ในการผ่านค่า)

การทำงานของซีพียู

การเรียกโปรแกรมย่อยผ่านเกตนี้ทำได้ทุกระดับความสำคัญ แต่เพื่อความสะดวกในการอธิบายและสมเหตุสมผลจะยกตัวอย่างเป็นโปรแกรมใช้งาน (ระดับความสำคัญต่ำ) เรียกใช้โปรแกรมย่อยในระบบจัดการ (ระดับความสำคัญสูง)

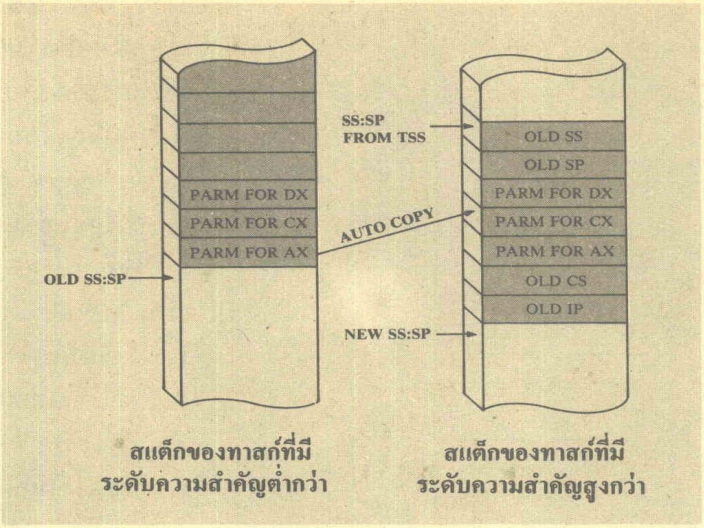
เมื่อโปรแกรมใช้งานใช้คำสั่ง CALL ผ่านเกต (เพียงคำสั่งเดียว) ซีพียูจะดำเนินการกับสแต็กดังรูปที่ 16 โดยเก็บค่า SS:SP ในสแต็กของระบบจัดการ และตามรูปสมมติว่าค่า Byte count เป็น 6 จึงถ่ายทอดข้อมูล 6 ค่า โดยสมมติว่าเป็นค่าของ DX, CX และ AX (จะเป็นค่าของอะไรก็ได้) เสร็จแล้วเก็บค่า CS:IP ของโปรแกรมใช้งานลงสแต็กของระบบจัดการเช่นกัน

ทางด้านแอดเดรส ซีพียูจะดำเนินการดังแสดงในรูปที่ 17 กล่าวคือ จะนำค่าซีเลคเตอร์ของคำสั่ง CALL ไปเปิดตาราง GDT ได้คอลล์เกตเดสคริพเตอร์มา นำค่า destination selector มาเป็นซีเลคเตอร์ตัวจริง ตรงจุดนี้เทียบได้กับใช้คำสั่ง

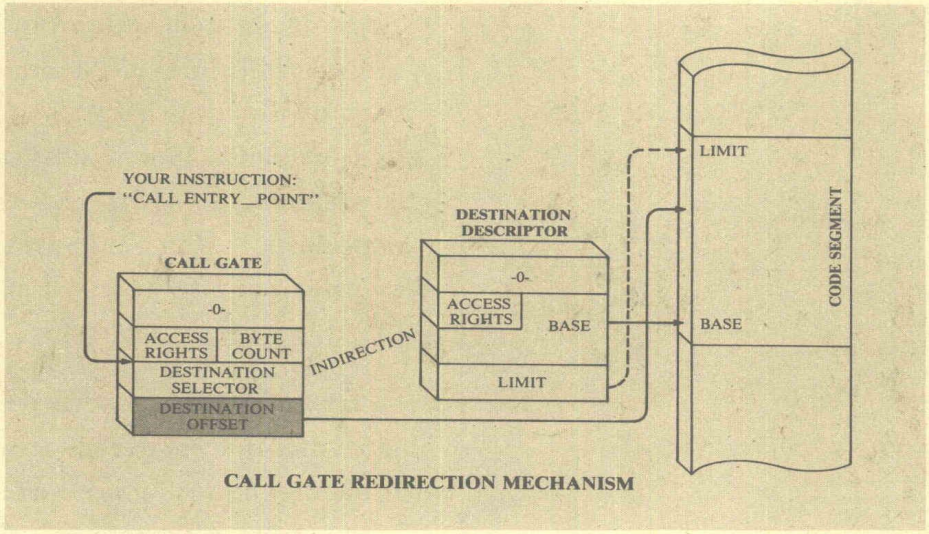
MOV CS, destination-selector

นั่นคือเป็นการให้ค่าเบสและลิมิตใน Destination descriptor (ตามรูป) แก่ CS cache ชี้ไปยังต้นเซกเมนต์ของ

รูปที่ 16 ▶
แสดงการเก็บค่าต่าง ๆ ในสแต็กของทาสก์ที่มีระดับความสำคัญสูงกว่า



รูปที่ 17
แสดงการทำงานของซีพียู เมื่อเรียกเข้าหาระบบจัดการ ▼



ระบบจัดการ เสร็จแล้วซีพียูจะโหลดค่า destination offset ในคอลล์เกตให้แก่ IP เท่ากับเป็นการกระโดดไปทำงานในโปรแกรมย่อยของระบบจัดการที่ต้องการ

เมื่อจบโปรแกรมย่อยในระบบจัดการจะมีการส่งกลับ(return) ซีพียูจะถอน CS:IP และ SS:SP คืนเท่ากับกลับไปโปรแกรมใช้งานโดยไม่ต้องผ่านเกต

จะเห็นว่าโปรแกรมใช้งานคือโปรแกรมที่มีระดับความสำคัญต่ำ จะเรียกใช้โปรแกรมย่อยในระบบจัดการคือโปรแกรมที่มีระดับความสำคัญสูงได้เท่าที่ระบบจัดการจะยินยอมให้ดำเนินการได้เท่านั้น เพราะโปรแกรมที่มีระดับความสำคัญต่ำจะไม่สามารถเข้าถึงตาราง GDT นั่นคือไม่สามารถสร้าง Call gate descriptor ได้ และระบบจัดการเป็นผู้กำหนดระดับความสำคัญของโปรแกรมใช้งาน

ขอให้ดูในรูปที่ 16 อีกครั้งครับ ทางด้านขวาของรูปแสดงข้อความว่าซีพียูได้ค่า SS:SP ของระบบจัดการจาก TSS (Task State Segment) ซึ่งจะกล่าวในหัวข้อต่อไป

การทำงานในแบบมัลติทาสก์

อาจมีบางท่านคอยว่าเมื่อไรผมจะกล่าวถึงการทำงานในแบบมัลติทาสก์เสียที ผมก็คอยครับแต่จำเป็นต้องกล่าวถึงเรื่องอื่นก่อน การทำงานในแบบมัลติทาสก์กับ 80286 ทำได้หลายแบบแล้วแต่ผู้เขียนระบบจัดการจะออกแบบ เช่น ใช้ Time interrupt ซึ่งจะแบ่งการทำงานของทาสก์ต่าง ๆ ให้ตามช่วงเวลาที่กำหนดถือเป็นแบบ time-slice

แต่ในหัวข้อนี้ผมจะกล่าวถึงเฉพาะกลไกที่ 80286 เตรียมไว้สำหรับการทำงานในแบบมัลติทาสก์ คือ การสับเปลี่ยนระหว่างทาสก์ที่เรียกว่าสวิตซ์ (switch) ซึ่งอินเทลให้ทำได้ด้วยการ CALL หรือ JUMP ข้ามเซกเมนต์ แต่ให้ CALL หรือ JUMP ผ่าน TSS

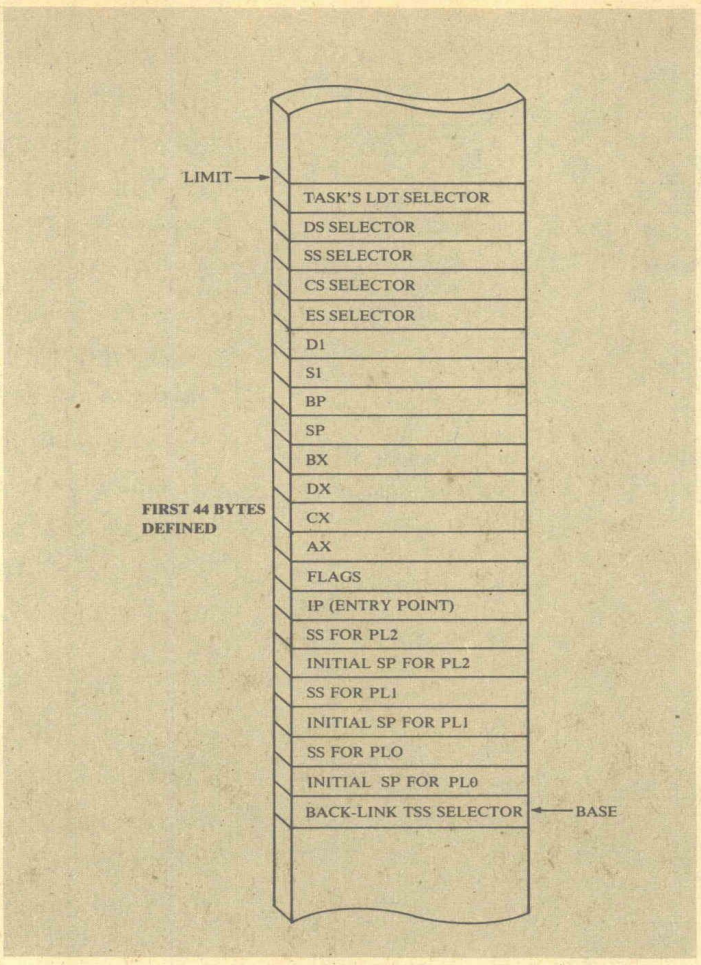
TSS (Task State Segment)

ในการ CALL ตามแบบ เช่น เมื่อเขียนโปรแกรมกับ 8086 คำสั่ง CALL จะเก็บ CS และ IP ลงสแต็ก แต่สำหรับ 80286 เมื่อทำงานในแบบมัลติทาสก์ ค่าที่เก็บไม่เฉพาะแต่ CS กับ IP เท่านั้น หากจะเก็บค่าใน General purpose registers ทุกตัว และค่าอื่น ๆ ซึ่งหากให้เก็บลงสแต็กจะสวิตซ์ได้ระหว่างสองทาสก์เท่านั้น อินเทลแก้ปัญหานี้โดยให้เก็บลง TSS

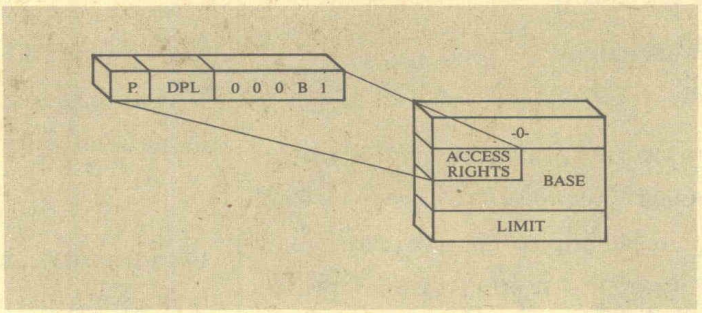
ค่าที่เก็บใน TSS ดังแสดงในรูปที่ 18 ซึ่งมีทั้งหมด 44 ไบต์ด้วยกัน การเตรียมข้อมูลในเซกเมนต์นี้ระบบจัดการจะต้องเตรียมเมื่อเริ่มต้น ซึ่งจะเป็นค่าเริ่มต้นที่จะให้แก่รีจิสเตอร์ต่าง ๆ ต่อไปด้วย และต้องเตรียมไว้ 1 TSS ต่อ 1 ทาสก์

เมื่อเตรียมเซกเมนต์ TSS แล้ว จะต้องเตรียมเดสคริพเตอร์ในตาราง GDT ด้วย เรียกว่า TSS descriptor เพราะมีลักษณะพิเศษแตกต่างไปจากเดสคริพเตอร์ธรรมดาดังแสดงในรูปที่ 19 คือตรง Access rights byte (ค่า B เริ่มต้นให้เป็น 0) โดยให้ base ชี้

รูปที่ 18
แสดงข้อมูลที่เก็บใน TSS



รูปที่ 19
เดสคริพเตอร์ของ TSS



มาที่ TSS และให้ค่า limit เป็น 43 และต้องให้ค่าซีเลคเตอร์ของ TSS descriptor ของทาสก์ในระบบจัดการแก่รีจิสเตอร์ TR ก่อนที่จะให้ทำงานในแบบโปรเทกโหมดเป็นการเริ่มต้นเท่านั้น ต่อไปค่าใน TR จะเปลี่ยนโดยอัตโนมัติ

การทำงาน

สมมติว่าผู้ใช้มีงานที่จะให้รันในแบบมัลติทาสก์อยู่ 3 ทาสก์ คือทาสก์ #1, #2, และ #3 ส่วนระบบจัดการก็มีหลายทาสก์เช่นกันให้ทาสก์ #A (ซึ่งพึ่งให้ค่าซีเลคเตอร์แก่รีจิสเตอร์ TR ในหัวข้อที่แล้ว) เป็นทาสก์ที่ควบคุมการทำงานทั้งหมดของระบบจัดการ

เมื่อจะรันโปรแกรมในแบบมัลติทาสก์ย่อมต้องมีกรรมวิธีที่จะให้ผู้ใช้ป้อนว่าจะรันโปรแกรมใดบ้าง สมมติว่าผู้ใช้ได้ป้อนแล้ว สำหรับการทำงานในแบบมัลติทาสก์นี้จะให้ทำในแบบ time-slice คือแบ่งช่วงเวลากันทำทาสก์ #A จะเดินวงจรนาฬิกาให้อินเตอร์รัพต์ตามช่วงเวลาที่ต้องการแล้วให้กระโดดคือ JUMP selector-1 (คือกระโดดด้วยค่าซีเลคเตอร์ไปยัง TSS descriptor #1)

ซีพียูจะอ่านค่าเดสคริพเตอร์เข้ามาตรวจที่ Access right byte ปรากฏว่าเป็น TSS descriptor ก็จะดำเนินการสวิตซ์คือเก็บค่าในรีจิสเตอร์ลง TSS #A ตามซีเลคเตอร์เดิมที่อยู่ใน TR เสร็จแล้วจะให้ค่า selector #1 แก่ TR ซึ่งจะเป็นการโหลดค่า base, limit และ access ใน TSS descriptor #1 ให้แก่ TR cache โดยอัตโนมัติด้วย (เมื่อให้ค่าแก่ TR ครั้งแรกก็ได้ให้ค่าแก่ TR cache ด้วย แต่ผมไม่ได้กล่าวถึงครับ)

จากนี้ซีพียูก็จะโหลดค่าในตาราง TSS มาให้แก่รีจิสเตอร์ต่าง ๆ ขอให้สังเกตว่ามีค่า IP ด้วย นั่นคือถ้ารันโปรแกรมค้างอยู่ตรงใดก็จะรันต่อ ณ จุดนั้น

เมื่อครบกำหนดเวลาวงจรเวลาจะอินเตอร์รัพต์เข้ามา ซีพียูจะไปทำงานในโปรแกรมตอบสนองอินเตอร์รัพต์ซึ่งอยู่ในระบบจัดการ โปรแกรมนี้จะสวิตซ์ให้ไปทำงานในทาสก์ต่อไปในกรณีนี้คือทาสก์ #2 ซีพียูจะดำเนินการดังที่ผ่านมาคือเก็บค่าในรีจิสเตอร์ขณะนั้นลงเซกเมนต์ TSS ตาม TR คือ TSS #1 แล้วให้ค่าซีเลคเตอร์ใหม่คือ selector #2 แก่ TR โหลดค่าต่าง ๆ ใน TSS #2 ให้แก่รีจิสเตอร์ ก็จะเป็นการทำงานในทาสก์ #2 จนกว่าจะมีสัญญาณอินเตอร์รัพต์เข้ามาอีกก็จะทำงานในทาสก์ #3 แล้ววนกลับไปทาสก์ #1 ใหม่

ทั้งหมดนี้เป็นการทำงานในแบบมัลติทาสก์อย่างคร่าว ๆ ครับ ส่วนที่ว่าเมื่อจบงานหนึ่ง ๆ แล้วจะทำอย่างไร หรือเมื่อผู้ใช้ต้องการเปลี่ยนงานใหม่จะทำอย่างไรเป็นเรื่องของการเขียนโปรแกรมครับ

สำหรับบิต B หมายถึง busy ที่ผมกล่าวไว้ว่าให้ค่าเริ่มต้นเป็น 0 นั้น หากซีพียูกำลังรันอยู่ในทาสก์ใดก็จะให้บิตนี้เป็น 1 เพื่อป้องกันไม่ให้สวิตซ์มาทำงานในทาสก์นี้ซ้อนกัน

ทาสก์เกต(Task gate)

กรุณาอย่าสับสนระหว่างคอลล์เกต(Call gate) ที่กล่าวในหัวข้อ "การเรียกเข้าหาระบบจัดการ" กับทาสก์เกตที่จะกล่าวในหัวข้อนี้ ทำงานคนละหน้าที่ แต่มีลักษณะงานคล้ายกันคือ สำหรับทาสก์เกตมีไว้เพื่อให้ทาสก์ที่มีระดับความสำคัญต่ำสามารถควบคุมการทำงานในแบบมัลติทาสก์ได้ คือสามารถสวิตซ์ได้

ทาสก์เกตเป็นเดสคริพเตอร์ตัวหนึ่งที่มีลักษณะพิเศษ ดังแสดงในรูปที่ 20 คือตรง Access rights byte ที่จะบอกซีพียูว่าเป็นทาสก์เกต ที่เขียนไว้เป็นค่า base จะต้องให้เป็นค่าซีเลคเตอร์ไปยัง TSS descriptor อีกทอดหนึ่ง ส่วนค่า limit ไม่ใช้

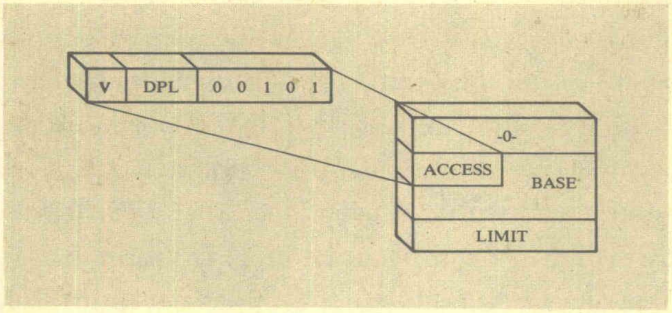
การใช้งานก็เช่นเดียวกับการสวิตซ์ที่กล่าวมา แต่ทาสก์ที่มีระบบความสำคัญต่ำจะกระโดดโดยตรงไปยัง TSS descriptor ไม่ได้ ต้องกระโดดผ่าน Task gate

โครงสร้างภายใน

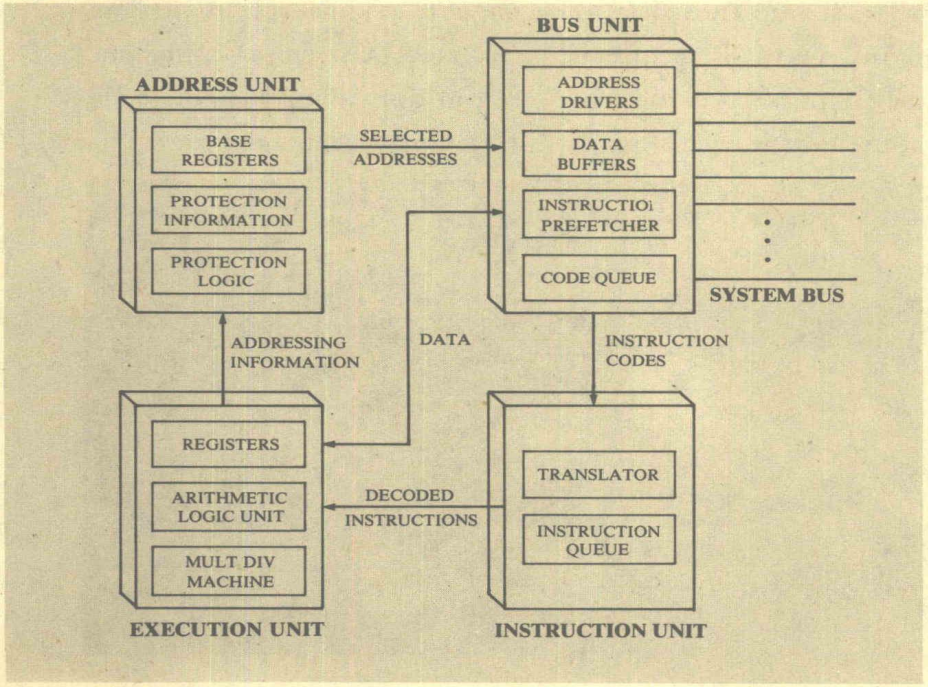
จะเห็นว่า 80286 มีขั้นตอนการทำงานมากมายแตกต่างไปจากซีพียูในยุคแปดบิตเดิมรวมทั้ง 8086 ด้วย โครงสร้างภายในจึงเปลี่ยนไปดังแสดงในรูปที่ 21 ในรูปได้แสดงถึง code queue ทั้งนี้เพราะ 80286 จะอ่านคำสั่งทุกรอบการทำงาน (ต่างจากสถาปัตยกรรมเดิมที่อ่านคำสั่ง ตีความ ปฏิบัติ แล้วจึงอ่านคำสั่งต่อไป) คำสั่งที่อ่านมานี้จะนำมาเข้าคิวไว้เพื่อรอการตีความต่อไป

เมื่อมีการกระโดด(JUMP CALL รวมทั้ง return) จึงต้องยกเลิกคำสั่งในคิว ข้อดีคือทำให้ซีพียูปฏิบัติตามคำสั่งต่อเนื่องได้เร็วขึ้น แต่ก็คงทำให้ผู้เขียนโปรแกรมลังเลที่จะเขียนโปรแกรมเป็นโมดูลขนาดเล็ก ๆ

รูปที่ 20 ▶ เดสคริพเตอร์ของทาสก์เกต



รูปที่ 21 ▼ แสดงโครงสร้างภายในของ 80286



กับ 80386

จุดอับของ 8086 ในเรื่องเซกเมนต์อินเทลพึ่งมาแก้ตกใน 80386 เพราะ 80386 เป็นซีพียูขนาด 32 บิต มีขาดาต้า 32 ขา และขาแอดเดรส 32 ขา ทำให้สามารถอ้างแอดเดรสได้ถึง 4 G ไบต์ (G หมายถึง Giga ประมาณพันล้าน) ได้อย่างต่อเนื่อง และสามารถอ้างแอดเดรสถึง virtual memory ได้ถึง 2× 8192×4G เท่ากับ 64T ไบต์ (T หมายถึง Tera ประมาณล้านล้าน)

ข้อแตกต่างที่สำคัญของ 80386 กับ 80286 เรื่องหนึ่งคือโหมดการทำงานซึ่งนอกจากจะมีเรียลโหมดกับโปรเทกโหมดดังกล่าวมาแล้ว ใน 80386 ยังมี Virtual 8086 mode ซึ่งจะแบ่งหน่วยความจำให้แก่โหมดนี้ 1 เมกะไบต์ โดยจะแบ่งเป็นกี่หน่วยก็ได้ และในแต่ละหน่วยอาจมีระบบจัดการของตนเอง เช่น แบ่งเป็น 2 หน่วย ให้หน่วยหนึ่งมี MS-DOS อีกหน่วยหนึ่ง นั้นมี CP/M พร้อมกันนี้ระบบจัดการหลักสมมติว่าเป็น OS/3 ก็จะทำงานอยู่ในโปรเทกโหมด

ความจริงแล้ว Virtual 8086 mode เป็นข้อต่อระหว่างเครื่อง 16 บิตกับ 32 บิตอย่างแท้จริง โปรแกรมที่มีอยู่เดิมสามารถนำมาใช้ได้โดยไม่ต้องปรับปรุงเลย ผมจึงไม่เข้าใจว่าทำไมไอบีเอ็มไม่เริ่ม PS/2 ที่ 80386